Client Meeting 4

```
#Drop Age and Cabin
df = df.drop(["Age","Cabin"],axis = 1)
print(df["Fare"].describe(),df["Embarked"].describe())
# Use mean for Fare and S for Embarked
```

```
count    1308.000000
mean       33.295479
std        51.758668
min         0.000000
25%         7.895800
50%        14.454200
75%        31.275000
max       512.329200
Name: Fare, dtype: float64 count     1307
unique       3
top          S
freq       914
Name: Embarked, dtype: object
```

```
df["Embarked"].fillna("S", inplace=True)
df["Fare"].fillna(df["Fare"].mean(), inplace=True)
```

- Age and Cabin had too many missing values so they were dropped
- Fare and Embarked na values filled in
- Categorical variable encoding:
  - *Mr, Mrs, Miss, and Master*
  - *Male and Female*
  - *Embarked*
- New feature: FamilySize = SibSp + Parch + 1(oneself)

```
# Extract Mr. Mrs. Miss and other titles:
df["Title"] = df["Name"].str.extract(" ([A-Za-z]+)\.")
df["Title"] = df["Title"].replace(["Ms", "Mlle"], "Miss")
df["Title"] = df["Title"].replace(["Mme", "Countess", "Lady", "Dona"], "Mrs")
df["Title"] = df["Title"].replace(["Dr", "Major", "Col", "Sir", "Rev", "Jonkheer", "Capt", "Don"], "Mr")
df = df.drop(["Name"], axis=1)
df["Title"].unique()
```

```
array(['Mr', 'Mrs', 'Miss', 'Master'], dtype=object)
```

```
# Encode the categorical variables
df["Sex"] = df["Sex"].map({"male": 1, "female": 0}).astype(int)
df["Embarked"] = df["Embarked"].map({"S": 1, "C": 2, "Q": 3}).astype(int)
df['Title'] = df['Title'].map({"Mr": 0, "Miss": 1, "Mrs": 2, "Master": 3}).astype(int)
```

	Pclass	Sex	SibSp	Parch	Fare	Embarked	Title	FamilySize
0	3	1	1	0	7.2500	1	0	2
1	1	0	1	0	71.2833	2	2	2
2	3	0	0	0	7.9250	1	1	1
3	1	0	1	0	53.1000	1	2	2
4	3	1	0	0	8.0500	1	0	1
...	...	...	...	...	...	...	...	...
886	2	1	0	0	13.0000	1	0	1
887	1	0	0	0	30.0000	1	1	1
888	3	0	1	2	23.4500	1	1	4
889	1	1	0	0	30.0000	2	0	1
890	3	1	0	0	7.7500	3	0	1

Implementing XGBoost

- Determined the best parameters based on gridsearch results
- Implemented XGBoost using the results
- Evaluated the cross-validation to ensure accuracy across folds.

```
# Use the best parameters
parameters = {
    'max_depth': [8],
    'n_estimators': [21],
    'learning_rate': [0.05]
}
# Best parameters:
    # 'max_depth': [8],
    # 'n_estimators': [22],
    # 'learning_rate': [0.05]
model_xgb = xgb.XGBClassifier(
    random_state=4,
)

model_xgb = GridSearchCV(
    model_xgb,
    parameters,
    cv=5,
    scoring='accuracy',
)

model_xgb.fit(X, y)
```

```
#XGBoost
parameters = {
    'max_depth': [5, 7,8, 9, 10, 11],
    'n_estimators': [5, 10, 15, 18, 20,22, 25, 30],
    'learning_rate': [0.01, 0.05, 0.06, 0.07,0.08, 0.09 ,0.1]
}

model_xgb = xgb.XGBClassifier(
    random_state=4,
)

model_xgb = GridSearchCV(
    model_xgb,
    parameters,
    cv=5,
    scoring='accuracy',
)

model_xgb.fit(X, y)

print(f'Best parameters {model_xgb.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: ' +
    f'{model_xgb.best_score_:.3f}'
)
```

Some takeaways

- Variable selections:
  - ~~Age and Cabin had too many missing values so they were dropped~~
  - Encoding Cabin, filling in Age NA values with mean
  - Fare and Embarked na values filled in
  - Title feature: *Mr, Mrs, Miss, and Master*
  - New feature: FamilySize = SibSp + Parch + 1(oneself)
- **More exploratory data analysis**
- **Carefully dealing with NA values:**
  - **If it's too much, it would be better to delete**
  - **When trying to fill in NA values with mean or mode, remember to use the entire dataset**
- **Double-checking after data cleaning**

```
[ ]  ### Encode cabin
     def encode_cabin(cabin):
         if pd.isna(cabin):
             return 0
         prefix = cabin[0]  # Get the first letter
         if prefix == 'A':
             return 1
         elif prefix == 'B':
             return 2
         elif prefix == 'C':
             return 3
         elif prefix == 'D':
             return 4
         elif prefix == 'E':
             return 5
         elif prefix == 'F':
             return 6
         elif prefix == 'G':
             return 7
         elif prefix == 'H':
             return 8
         else:
             return 9

     # Apply the function to the Cabin column
     train_data['Cabin'] = train_data['Cabin'].apply(encode_cabin)
     test_data['Cabin'] = test_data['Cabin'].apply(encode_cabin)
```

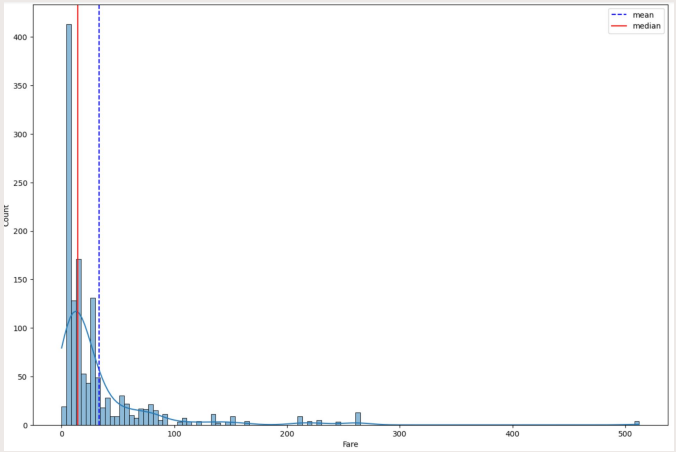
```
[7]  # Clean NAs for "Embarked", "Fare", "Age"
     df["Embarked"].fillna("S", inplace=True)
     df["Fare"].fillna(df["Fare"].mean(), inplace=True)
     df["Age"].fillna(df["Age"].mean(), inplace=True)
```

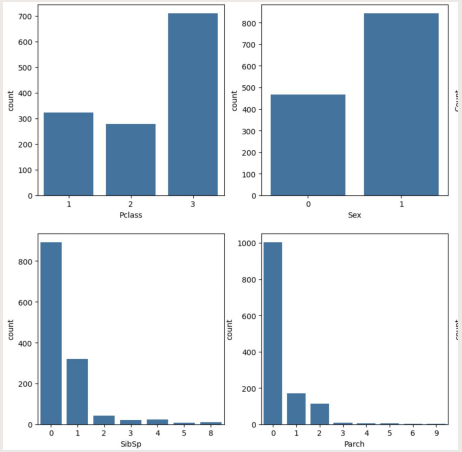
```
     # Clean "Title"
     df["Title"] = df["Name"].str.extract(" ([A-Za-z]+)\.")
     df["Title"] = df["Title"].replace(["Ms", "Mlle"], "Miss")
     df["Title"] = df["Title"].replace(["Mme", "Countess", "Lady", "Dona"], "Mrs")
     df["Title"] = df["Title"].replace(["Dr", "Major", "Col", "Sir", "Rev", "Jonkheer", "Capt", "Don"], "Mr")
     df = df.drop(["Name"], axis=1)
     df["Title"].unique()
```

```
array(['Mr', 'Mrs', 'Miss', 'Master'], dtype=object)
```

```
[9]  # Encode the categorical variables
     df["Sex"] = df["Sex"].map({"male": 1, "female": 0}).astype(int)
     df["Embarked"] = df["Embarked"].map({"S": 1, "C": 2, "Q": 3}).astype(int)
     df['Title'] = df['Title'].map({"Mr": 0, "Miss": 1, "Mrs": 2, "Master": 3}).astype(int)
```

Some takeaways






```
parameters = {
    "n_estimators": [5, 10, 15, 20, 25, 30],
    "max_depth": [3, 5, 6, 7, 8, 11],
}

model2 = RandomForestClassifier(random_state=1)

model2 = GridSearchCV(
    model2,
    parameters,
    cv=5,
    scoring='accuracy',
)

model2.fit(X, y)
# Sorry if the X and X_train confuses you

print('-----')
print(f'Best parameters {model2.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: '+ \
    f'{model2.best_score_:.3f}'
)
print('-----')
```

Random Forest w/o parameter tuning

CV score: .827 Score 7799

```
model2 = RandomForestClassifier(n_estimators=15, max_depth=7, random_state=1) #Sangjun's X and y
# model2 = RandomForestClassifier(n_estimators=10, max_depth=6, random_state=1)
model2.fit(X, y)
predictions = model2.predict(X_test)
```

- Among different training models:
  - Generally **XGBoost and Random Forest** performs better than logistic regression
    - Better capability to handle imbalanced data
  - Compared to random forest, XGBoost is more sensitive with different variable selection.
    - Random forest: more robust to noisy data and small features

submission(rf_com) (3).csv Complete · 14h ago	**0.77751**
titanicsubmission(xgb_com) (3).csv Complete · 14h ago	**0.76794**
titanicsubmission(xgb_com) (2).csv Complete · 15h ago · xgb_com w/o family & cabin	**0.76076**
submission(rf_com) (2).csv Complete · 15h ago · rf com w/o family & cabin	**0.77511**
submission(rf_com) (1).csv Complete · 15h ago · rf combination w/o family	**0.77751**
titanicsubmission(xgb_com) (1).csv Complete · 15h ago · xbg combination w/o family	**0.75837**
titanicsubmission(xgb_com).csv Complete · 15h ago · xgb_com	**0.76315**

XGBoost

CV score: .838 Score: .78707

```
# Best parameters:
    # 'max_depth': [8],
    # 'n_estimators': [22],
    # 'learning_rate': [0.05]
```

Random Forest w/ parameter tuning

CV score: .832

Score: .77751

Score (Anshi's X and y): .607

Logistic Regression

CV_score = .806 Score = .758